Express Mail Label Number:
EL549238284US
Date of Deposit: October 31, 2000

Method and Apparatus for Distributed Application Acceleration

FIELD OF THE INVENTION

The present application relates to the field of computer networks and distributed applications. It is more particularly directed to applications that are operational on the Internet using a system of Web-browsers and Web-servers.

BACKGROUND OF THE INVENTION

Currently, access over the Internet to Web-based applications is provided by having a Web-browser connect directly over a network of routers to a Web-server that maintains static content in data files, and composes dynamic content by executing programs, typically cgi-bin scripts or Java servlets. However, during periods of congestion due to traffic patterns on the Internet, this arrangement results in poor response times for the end client. The situation is typically worse the farther the client is located from the Web-server and the greater the number of intermediary routers involved in the network connection.

One way to improve application response time, reliability, and availability is to distribute the applications to proxy servers located closer to the client browsers. Distribution of content is used to improve the performance of the network by means of proxies within the network that cache pages.

The simple caching approach works well for data that is static and unchanging, e.g. images, video clips, etc. A proxy server is deployed within the network in many different ways. Some of the common ways include using a proxy server as a reverse proxy, where the proxy server is located closer to the web-server it is proxying for; as a forward proxy, where the proxy server is located closer to the browser or the client applications; or is as other auxiliary servers which may be located elsewhere within the network. The proxy server usually provides information to the browser on behalf of a backend server. The browser may contact the proxy server due to a variety of reasons e.g., because it has been explicitly configured to do so, or because the domain name server gives it the location of a proxy server instead of the backend web-server, or because a network operator or backend web-server operator has configured the network to send requests from the browser to the proxy server in a transparent fashion.

However, the techniques of caching that are commonly deployed in the current Internet do not work well with a large portion of web-accessible content. Data that is personalized to a client, or data that is generated by invocation of programs like cgi-bin scripts or servlets can not be readily cached at the proxies. For a server offering electronic services over the Internet, non-static data forms a significant portion of their overall data content. It would be advantageous to have a scheme whereby such dynamically generated content, and web-centric applications can also benefit from the presence of proxies.

As in the case of caching of static data, it is highly desirable that the caching of applications be done so that the administrative and operation control of the data/application resides with the original server, rather than with at the proxy server. A solution is needed which accelerates applications while still providing the administrative control of the application to the original server, rather than the proxy server.

SUMMARY OF THE INVENTION

Accordingly, an aspect of the present invention presents methods and apparatus by which to accelerate execution of Web front-ended applications by means of executing them at proxies located closer to the client browsers.

Another aspect of the present invention presents methods and apparatus for a proxy server which provides an execution environment for acceleration of Web front-ended applications.

Another aspect of the present invention presents methods and apparatus for a backend server which provides an execution environment for acceleration of Web front-ended applications.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other aspects, objects, features, and advantages of the present invention will become apparent upon further

consideration of the following detailed description of the invention when read in conjunction with the drawing figures, in which:

Fig. 1 is a block diagram showing an example of an application acceleration infrastructure, and their manner of interaction in accordance with the present invention;

Fig. 2 is a block diagram of an example showing steps for a method of application acceleration in accordance with the present invention;

Fig. 3 is a block diagram of an example showing steps taken by a proxy server in executing a requested service for an end client in accordance with the present invention;

Fig. 4 is a block diagram of an example showing steps for determination of an appropriate set of programs to run in response to a particular end client service request in accordance with the present invention;

Fig. 5 is a block diagram of an example showing structure of an information record, used in a solution to the distributed application acceleration problem in accordance with the present invention;

Fig. 6 is a block diagram showing an example of a structure of a distributed system to implement a solution to the distributed application acceleration problem in accordance with the present invention;

Fig. 7 is a block diagram showing an example of a structure of a proxy server, used in a solution to the distributed application acceleration problem in accordance with the present invention; and,

Fig. 8 is a block diagram showing an example of a structure of a backend server, used in a solution to the distributed application acceleration problem in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 shows an example of the components of an application acceleration infrastructure. A communication network 101 is used to interconnect client devices containing Web-browsers like 102 and 103 to a backend web server 104 that provides Web-based applications. Major portions of the application code are moved onto proxy servers like 105 and 106 that are in closer proximity to the respective end client devices. Communications is maintained between the proxy servers 105 and 106 and the backend server 104 so that the backend server can continue to exercise administrative control over the distributed portions of the applications code. Communications is also maintained between the end client devices 102 and 103 and the backend server 104 so that the latter can continue to provide applications services that are not readily distributable.

Fig. 2 delineates the steps that comprise a method of application acceleration. In response to a client service request 201, a Wide Area Load Balancing module determines

the appropriate proxy server to which to direct the request based upon current network performance characteristics and the location of the end client client device as in 202. The proxy server then determines the set of programs that it needs to run in order to satisfy the end client request as in 203. These may be already resident at the proxy server if they had been needed to satisfy previous requests from other users, or if they had been deployed to the proxy server in anticipation of end client requests for basic or popular applications. In any of these cases, they are resident at the backend server. The proxy server obtains the data and application code appropriate for fulfilling the service request as in 204, and executes the indicated functions for the end client on behalf of the backend server as in 205.

Fig. 3 is a block diagram that shows an example of steps taken by a proxy server in executing a requested service for an end client. The response from an application is often tailored to a particular end client or class of end users, depending upon organizational affiliations or other criteria. The proxy server thus obtains information from the backend server regarding which execution parameters to employ in satisfying this specific end client request as in 301. The application logic is then executed as in 302, and any administrative information or error messages resulting from the application execution are logged and sent to the backend server as in 303. In order to simplify systems management and provide support for problem tracking and diagnosis, the backend server maintains responsibility for the application logic. This allows the proxies to be shared effectively by many back end servers without increasing

their complexity. The results of the execution of the application logic are then sent to the end client as in 304.

Fig. 4 is a block diagram of steps for a determination of an appropriate set of programs to run in response to a particular end client service request. The request indicates a target location within the backend server for the application logic, typically in the form of a Uniform Resource Locator (URL). The proxy server maintains data on application programs that it represents, and determines what service is being requested as in 401. The appropriate proxylet record is then retrieved as in 402. The proxylet record contains a data field specifying the set of programs needed in order to properly satisfy an end client service request, and this is read by the proxy server as in 403.

Figure 5 shows the structure of a proxylet record used in order to determine the set of programs that are to be executed at a proxy when a request is received from a client. The proxylet record 501 includes several fields. The RequestedURL field 503 contains the URL which the client was requesting. The ExecuteURL field 505 contains the name of a URL which will be executed at the local proxy in order to support this request. The location where the compute code required to run ExecuteURL is contained in the field CodeLocation 507. The ParameterList field 509 contains any parameters which would be passed to the program in order for it to be executed. The contents of the ParameterList field 509 may be same for different proxies within the network, or they may be different for multiple proxy servers. The ExpirationTime field 511 contains the date until which this proxylet record may be considered valid. Once the time

specified by field 511 has expired, a proxy server would need to retrieve a new proxylet-record from the backend server. The LoggerURL field 513 identifies a location where the error messages and diagnostic output of the proxylet are provided. Typically, the loggerURL would identify a location on the backend server. The codeVersion field 515 contains the time when the program set identified in the field codeLocation 505 was last modified. Other fields may also be included within the proxylet-record as used in different embodiments of the invention.

As an example, let us consider a request from a client which is targeted to the location

http://main-server.com/servlet/program1.

This request is delivered to the proxy server running at the machine proxy-server.com. The proxylet record for this request might contain the fields of requestedURL field being

http://main-server.com/servlet/program1,

with executeURL field being /servlet/proxy-program1, the codeLocation field being

http://main-server.com/proxylet/prox-program1,

the ParameterList being empty, and the LoggerURL being

http://main-server.com/servlet/logger,

the expirationTime field being 30000 seconds after a reference date such as 1/1/1970, and the codeVersion field being 29500 seconds after a reference date such as 1/1/1970.

When such a request is received at the machine proxy-server.com, the machine checks if it has a cached entry corresponding to the proxylet-record where the requestedURL matches the request being received. If it does, it checks the expirationTime field to ensure that the record needs to be updated. It then checks to see if it has the program identified by the executeURL installed locally at the proxy-server. If it does, then it runs the program passing to it any parameters contained in the parameterList field. If the proxylet-record is not found, or if the current time is greater than the time specified in the expirationTime field, the proxy-server contacts the main-server to obtain a fresh copy of the proxylet-record prior to executing the steps outlined above.

The schemes as described above can be seen as a distributed system that achieves acceleration of applications by distributing their execution. An apparatus 601 that implements the distributed system is shown in Figure 6. Figure 6 includes three distributed components, a wide-area load balancer 603, a application distributor 605, and a base class library 607. The wide area load-balancer 605 is a distributed module which collects performance statistics about the network, and determines the most appropriate place where a request should be sent out to. Since these components are distributed components, the connectivity information among them is not shown in the Figure. However,

the connectivity information will become clearer from the description provided below.

The Wide Area Load Balancer 603 is a component responsible for distributing client requests to different proxy servers within the network. It can be implemented in a variety of manners. One common way to implement it is by means of a modified domain name server. The domain name server, usually abbreviated to DNS server, is the application in the network responsible for mapping machine names to IP addresses. A modified domain name server can return an IP address which corresponds to an appropriate proxy server when a client requests an address for the backend server. The appropriate proxy server is determined on the basis of the current network performance characteristics and the location of the client.

An alternative implementation of the wide area load balancer includes a module within the backend server that is responsible for redirecting requests to the appropriate proxy server. Such a redirection module might be implemented as a plug-in module among a variety of web-servers such as Apache, NetScape or Microsoft IIS server, which are commonly in use in the industry. The module would look at a table of redirection rules, which specify how requests coming from specific client IP addresses should be dispatched, and use this information to determine the appropriate proxy-server to which the request should be dispatched. The selection of the proxy-server can be based on other criteria included in the rule, e.g. The resource (URL) being requested by the client, or a cookie which is contained within the client's request.

Another embodiment of the wide area load balancer uses a stand-alone http server which provides the same functionality as that of the module described above. The http server implements the ability to direct requests to proxy-servers, or to another server operation locally at the site with the stand-alone http server, which services requests that need to be performed locally.

The application distributor 605 is responsible for ensuring that the set of programs that need to be executed at the proxy server are indeed available at that server. There are many embodiments of such an application distributor which will be useful to those skilled in the art.

One useful embodiment of an application distributor uses a program which keeps track of all the programs and data that is available at a main-server, and maintains a replica of those program and data at the proxy server. Such an distributor will push the changes that occur at the main-server out to the proxy servers in order to maintain this consistency of program and data.

Another embodiment of an application distributor uses a program that runs at the proxy-server and caches a copy of programs and data from the main-server when requests that would cause execution of those programs are received at the proxy-server. The data that the programs need to execute is also retrieved as needed, and cached at the proxy-server. The programs and data are both cached on demand.

Yet another embodiment of an application distributor uses a program that employs both of these techniques. Some of the programs which are most often used are pushed out to all of the proxy-servers while other programs are cached on demand at the proxy-server.

The class library 607 is a set of programs that exists at all the proxy-servers and the backend servers. It contains a collection of classes that enable many functions to occur. One of the classes contained in the library identifies the set of programs that are capable of executing at the proxy-server. All such programs are derived from a specific class proxylet, and the fact that these programs are subclassed from the class proxylet is used to validate that the program can execute at the proxy-server. Yet another class provided in the class library is the Logger class, which allows the output and error messages generated by the program executing at the proxy-server to be copied to the main-server for purposes of logging and diagnostics. Yet other set of classes allow for the caching of different types of application data. Instances of these include the programs for caching queries made to a directory or a database, programs for caching records in a database, as well as programs for caching files.

The components of the distributed architecture shown in Figure 6 are various proxy-servers and the backend server. A proxy-server which is such a component in this solution is shown in Figure 7. The proxy-server 701 includes a Cache Manager 703, a set of cached information records 705, a set of cached programs 707, and a set of cached data 709. The Cache Manager 703 is responsible for managing and updating

the different types of caches, namely the set of cached information records 705, the set of cached programs 707 and the set of cached data 709. The cache manager maintains all of these caches in an appropriate manner. The set of cached information records 705 contains information about the cached programs that are cached locally in the set of cached programs 707. The format of information contained in the information record is similar to that of the proxylet record 501 shown in Figure 5. The Cache Manager 703 utilizes the services of an application distribution module 605 in order to ensure the consistency and coherence of the different sets of caches. The set of cached data programs 709 is maintained current by using data caching techniques that are well-known to those versed in the art.

Figure 8 shows the structure of a backend server which would respond to the proxy server shown in Figure 7 and provides the other part of the infrastructure for application acceleration. The backend server 801 includes a traditional web-server 803, a set of programs to be downloaded to proxy servers 805, a set of local programs 807, and a set of operational programs 809. The web-server 803 provides the means by which a proxy server can gain access to the set of programs 803, 805 and 807. The set of downloadable programs 803 are transferred to a proxy server if it makes a request for them. All or a subset of the programs may be transferred to the proxy server. The set of local programs 807 provides a means by which a proxy server can execute some parts of the processing at the backend server itself. As an example, a proxy server may want to execute programs related to updating databases only at the backend server. The set of operational programs 809 provides a means by

which a proxy-server can provide diagnostics and management information to the backend server. An example of an operational program 809 would be a logger servlet that can obtain logging messages generated by programs executing at the proxy server.

In some embodiments of the backend server, the web-server may incorporate an ability to redirect client requests to other servers. This would be an instance of the application distribution module. In other embodiments, the backend server may rely upon the domain name service to do such redirections. The backend server as described in Figure 8 and a set of proxy servers as described in Figure 7 together provide the infrastructure for distributed application acceleration.

It is noted that the present invention can be realized in hardware, software, or a combination of hardware and software. A tool according to the present invention can be realized in a centralized fashion in one computer system, or in a distributed fashion where different elements are spread across several interconnected computer systems. Any kind of computer system - or other apparatus adapted for carrying out the methods described herein - is suitable. A typical combination of hardware and software uses a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein. The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which -

when loaded in a computer system - is able to carry out these methods.

Computer program means or computer program in the present context includes any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either conversion to another language, code or notation; and/or reproduction in a different material form.

It is noted that the foregoing has outlined some of the more pertinent objects and embodiments of the present invention. This invention may be used for many applications. Thus, although the description is made for particular arrangements and methods, the intent and concept of the invention is suitable and applicable to other arrangements and applications. It will be clear to those skilled in the art that modifications to the disclosed embodiments can be effected without departing from the spirit and scope of the invention. The described embodiments ought to be construed to be merely illustrative of some of the more prominent features and applications of the invention. Other beneficial results can be realized by applying the disclosed invention in a different manner or modifying the invention in ways known to those familiar with the art.